**고급소프트웨어실습1: 5주차 과제**

**20171672 이정원**

**프로그램 소개**

GUI 소프트웨어를 사용하여 임의의 확률 밀도 함수 px(x)를 설계하고, px(x)를 따르는 확률 변수 X를 시뮬레이션 해 주는 난수 수열 X0, X1, X2, X3, ... 를 확률적으로 올바른 방법으로 발생시킨 후, 자신이 올바르게 난수를 생성하였는지 통계적으로 확인하는 프로그램이다.

**프로그램 구동 방법**

원하는 곡선 샘플을 sampling\_table.txt로 프로젝트와 같은 폴더에 저장하여 program2\_1을 통해 정보를 읽고 정규화 과정을 거친다. 정규화된 값들의 적분값을 콘솔로 보여 줄 것이다.

program2\_2는 이 곡선 분포를 따르는 난수를 발생시키는 프로그램이다. 원하는 난수 개수를 입력하면 random\_event\_tabe.txt에 저장된다.

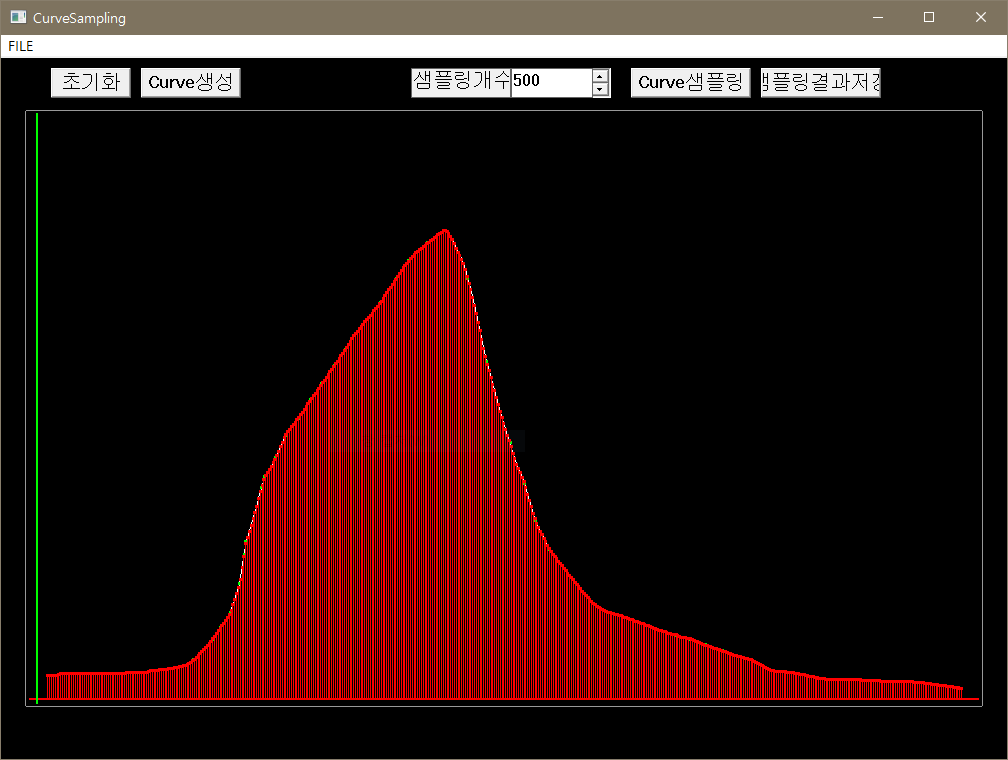
program2\_3은 앞서 생성한 난수들이 올바르게 생성된 것인지 histogram.txt에 결과가 저장되는 프로그램이다. 이 과정이 끝나면 function2\_1(), program2\_2\_a(), program2\_2\_b()가 호출된다.

function2\_1()에서는 지수 분포를 따르는 난수를 발생시키는 프로그램이다. 원하는 난수 개수를 입력하면 각 람다에 따른 평균값과 분산값의 이론값과 실험값이 출력된다.

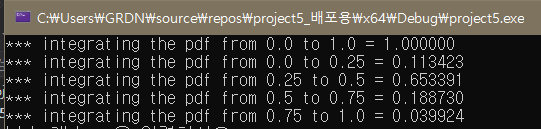
program2\_2\_a(), program2\_2\_b()는 Bisection Method, Secant Method, Newton Method을 이용해 곡선 분포를 따르는 난수를 발생시키는 계산 시간을 출력하는 프로그램이다. 이때 발생되는 난수 개수는 모두 1000이다.

**실습 문제 2-1**

sampling\_table.txt 파일에 저장되어 있는 함수 정보를 입력받아 정규화를 거친 뒤 구분구적법을 이용해 적분값을 구하는 문제이다. 정규화된 정보는 pdf\_table.txt에 저장한다. 아래는 실험에 이용한 곡선이다.



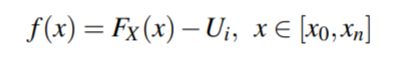
sampling\_table.txt에서 읽은 정보는 각각 n1, h, 배열 x, y에 저장하였다. x의 최대값과 최소값을 구한 뒤 정규화 계산을 한 후 new\_x에 저장한다. y의 정규화는 구분구적법, 즉 사다리꼴 넓이를 모두 더한 값을 sum에 저장한 뒤 new\_y에 저장할 때 각 y값에 sum을 나눠 준다. 이 정규화된 값들이 적분했을 때 각 범위에 대해 대략적으로 맞는 값이 나오는지 확인하는 문구를 출력한다. 출력 결과는 다음과 같다.



[0.0, 1.0]의 적분값은 1이며, 다른 값들에 비해 [0.25, 0.5] 구간의 적분값이 큰 것을 보아 곡선의 모양과 대략적으로 일치한다는 것을 확인할 수 있었다. 또한, 부분 적분한 값들을 모두 더하면 0.99 정도로 1에 가까워 계산이 알맞게 된 것으로 확인하였다.

**실습 문제 2-2**

생성할 난수 개수를 입력받아 이 곡선 분포를 따르는 난수를 생성해 파일에 출력하는 문제이다. 우선 무작위로 [0.0, 1.0] 사이의 값을 가지는 난수를 iseed에 생성한다. 곡선 분포를 따르는 난수를 생성하기 위해서는 누적 분포 함수의 역함수 값을 알아야 하는데, 그렇게 되면 다음과 같은 식의 근을 구하는 문제로 귀결된다.



새로운 함수 fx(double xn, double iseed, double \*x, double\*y, int n)를 선언하여 pdf\_table.txt의 값들을 읽어와 적분값을 구하는 함수를 구현하였다. Bisection Method를 이용해 이 함수의 근을 구하고, 정확한 근이거나 종료 조건에 도달했을 경우 출력 파일에 결과를 저장한다. 이 과정을 입력받은 난수 개수만큼 반복한다.

**숙제 2-1**

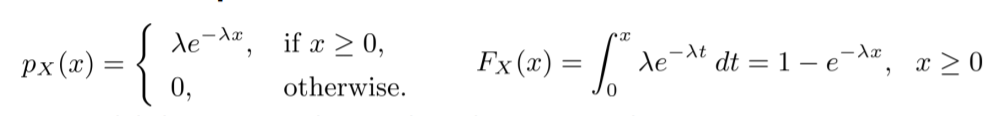
숙제 2-1 항목은 program2\_3.cpp 에서 void function2\_1() 라는 새로운 함수를 선언해 주어 숙제 코드를 작성하였다. function2\_1() 함수는 실험을 위한 난수 개수 nr을 입력받은 뒤, 난수의 개수만큼 0에서 1 사이의 범위를 가진 난수를 생성하여 U라는 배열에 할당한다.

1. **지수 분포의 인자 λ (>0)에 대해 확률 변수 X의 기대값이 E[X] = 1/λ이고 분산이 Var[X] = 1/λ^2이라는 사실을 고려하여, 세 가지 서로 다른 λ 값을 설정하라.**

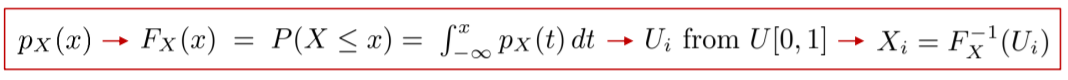
3개의 λ를 각각 변수 ramda\_a, ramda\_b, ramda\_c로 지정하였다. 각각 0.5, 1, 1.5로 선언하였다.

1. **자신이 정한 λ에 해당하는 지수 분포 각각에 대하여, inversion 방법을 사용하여 자˙ 신˙ 이˙ 정˙ 한˙ 충˙ 분˙ 히˙ 큰˙ 개˙ 수˙ 만˙ 큼˙ 이 지수 분포를 따르는 난수를 생성하라.**

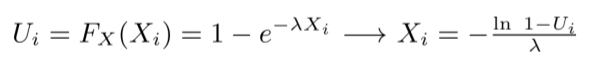
지수 분포(exponential distribution)의 확률 밀도 함수와 누적 분포 함수는 다음과 같다.



지수 분포를 따르는 난수를 발생시키기 위해서는 확률 밀도 함수와 누적 분포 함수를 알아야 한다. Inversion Method를 이용하면, 다음과 같이 확률 분포를 따르는 난수 수열을 정의할 수 있다.



여기서 U는 [0, 1] 구간의 값을 가지는 난수 수열이다. 위 식에 의해 지수 분포를 따르는 난수 수열의 정의는 다음과 같다.

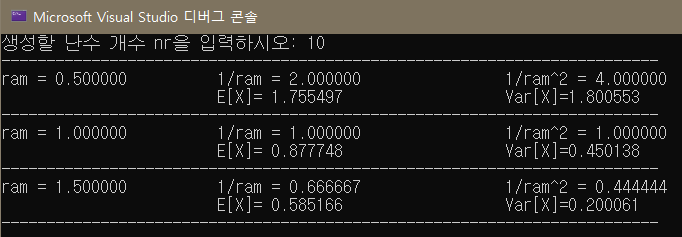


위 식을 참고하여 지수 분포를 따르는 난수를 function2\_1()에서 생성하여 X라는 배열에 저장하였다.

1. **자신이 생성한 난수들을 사용하여 평균값과 분산값을 구한 후 이론적인 값들과 얼마나 일치하는지 분석하라.**

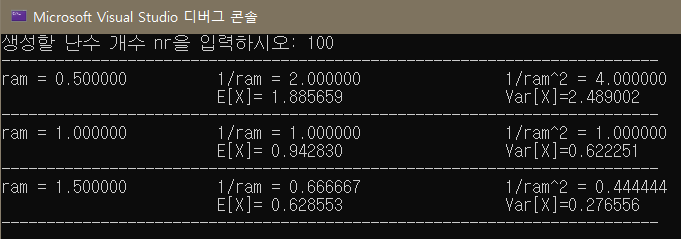
각 λ에 대하여 각각의 평균과 분산을 구한 후 이론적인 평균값(1/λ)과 분산값(1/λ^2)을 비교하기 위해 모두 출력하였다. 출력한 결과는 다음과 같다.

<입력한 난수 개수 nr이 10일 경우>



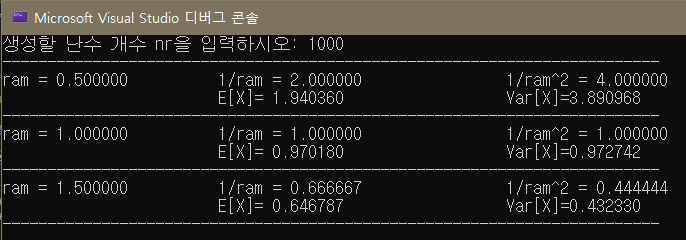
10과 같이 난수 개수가 작은 경우에는 이론값과 실험값의 차이가 많이 나는 것을 확인할 수 있다.

<입력한 난수 개수 nr이 100일 경우>

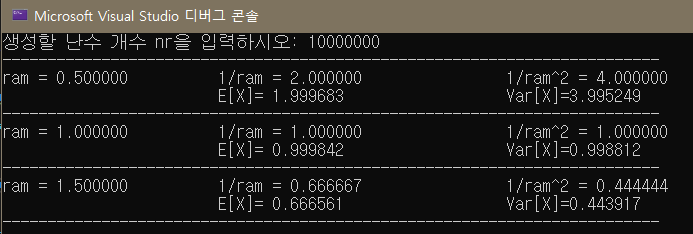


100일 경우 10일 경우와 비교했을 때 차이가 줄어든 것을 확인할 수 있었다.

<입력한 난수 개수 nr이 1000일 경우>



<입력한 난수 개수 nr이 10000000일 경우>



1000일 경우, 10000000일 경우를 확인해 본 결과 nr이 커질수록 이론값과의 차이가 줄어드는 것을 확인할 수 있다. 특히 1000000인 경우는 난수 개수가 100만 개 정도는 되어야 이론값과 0.0003정도밖에 차이가 안 나는 등 이론값들과 충분히 일치하게 되는 것으로 판단하였다.

**숙제 2-2**

**(i-ii) 실습 시간에 작성한 코드 (프로그램 2-2)를 잘 가다듬어 문제없이 그리고 효율적으로 수행되도록 수정하라. (프로그램 2-2(a)). 주어진 확률 밀도 함수에 대하여 자신의 코드가 올바르게 난수를 생성하는지 통계적으로 확인하라. 이를 위하여 프로그램 2-3을 작성한 후 이를 활용하라.**

**[프로그램 2-2(a)]**

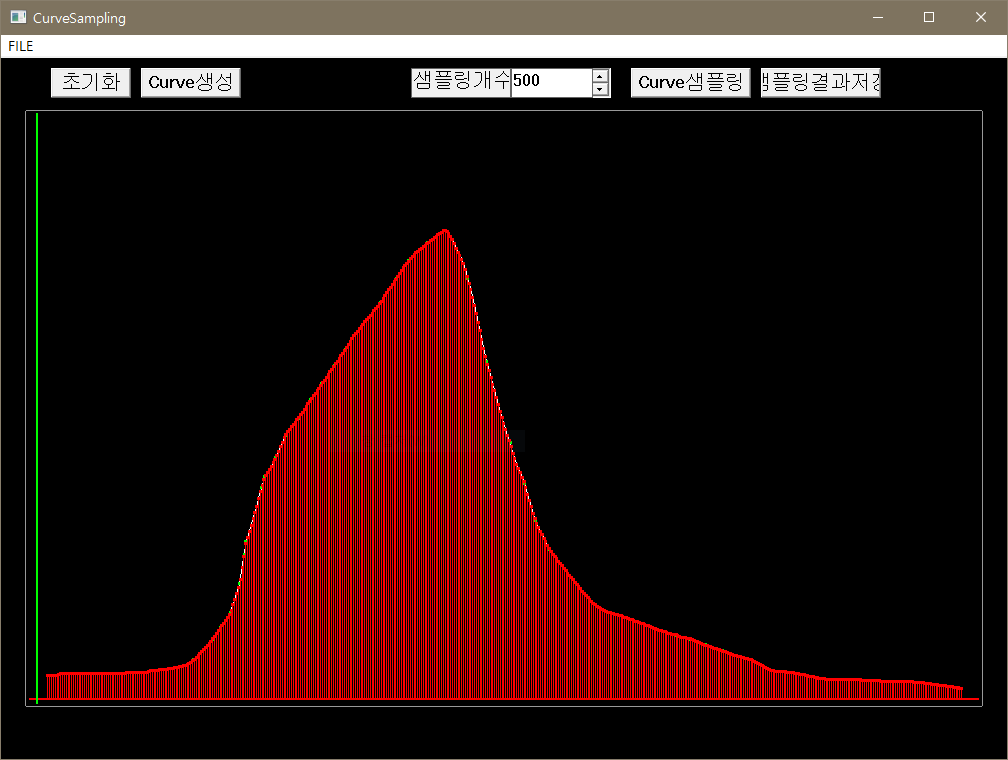
실습에서 사용했던 프로그램 2-2에서 적분을 구하는 함수 계산식(double fx(xn, seed, x, y, n))이 잘못되어 생성된 난수들 중 일부가 같은 숫자(예시: 0.999999996362)가 반복되는 문제가 있었다. 적분 구하는 함수 계산식에서 함수의 인덱스를 잘못된 변수로 지정해서 일어난 문제였다. 이를 올바르게 고쳤다.

**[프로그램 2-3]**

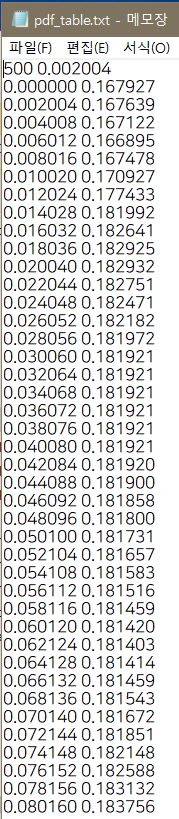
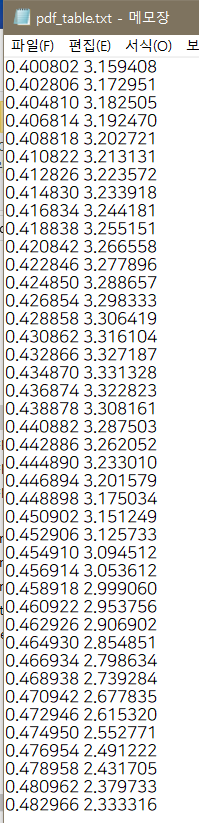
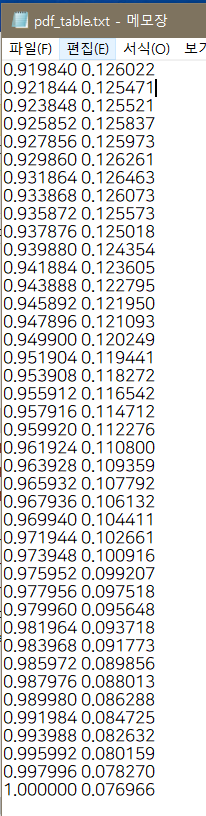
프로그램 2-3에서는 pdf\_table.txt와 random\_event\_table.txt 파일에서 확률 밀도 함수와 생성한 난수 정보를 읽어들여 변수와 배열을 할당하여 저장한다. 특히, random\_event\_table.txt에서 읽어온 난수의 개수는 nr에, 난수들은 배열 xr에 할당하여 저장하였다. 우선 배열 xr의 원소들을 오름차순으로 정렬한 뒤, [0.0, 1.0] 구간을 500개(임의로 정한 숫자)로 나누어 각 난수가 속해 있는 구간을 찾는 과정을 반복하였다. 해당 구간에 난수가 속해 있을 경우, 해당 구간의 난수 개수 변수(xr\_count)를 1 늘리고, 없으면 해당 구간의 반복문을 빠져나와 다음 구간으로 넘어간다. 다음 구간으로 넘어가기 전, histogram.txt에 해당 구간의 정보들을 출력하였다. (해당 구간, 해당 구간의 난수의 개수, 해당 구간의 난수의 개수/전체 난수 개수)

프로그램이 올바르게 작동하는지 확인하기 위해 다음의 실험 데이터를 사용하였다.

<실험에 이용한 곡선>

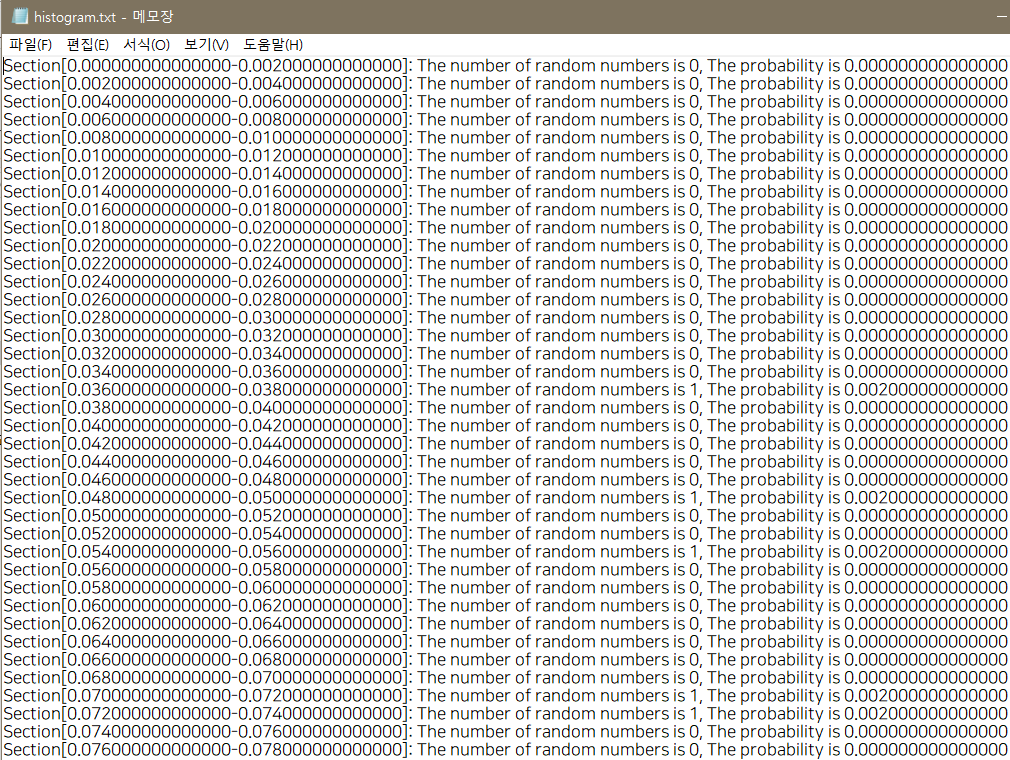


<pdf\_table.txt>

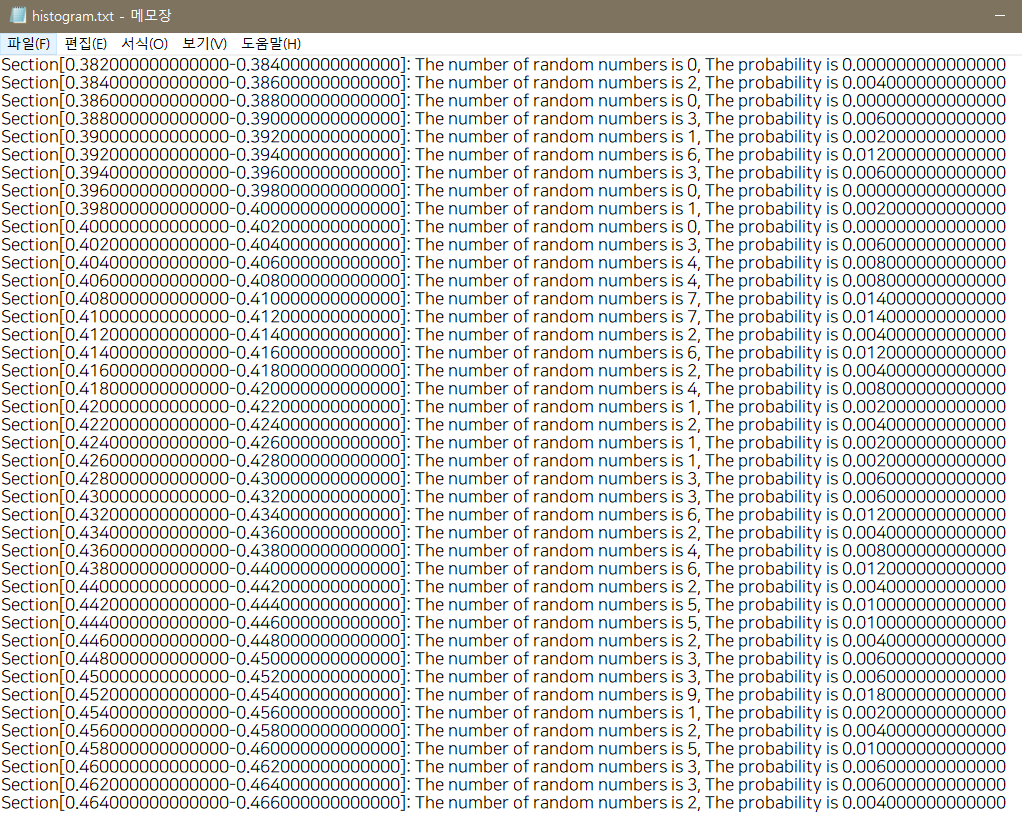
...  ... 

커브의 모양처럼 처음에는 정규화된 y값이 작으나, 중간 구간에서는 y값이 갑자기 커지고, 끝구간에서는 y값이 다시 작아지는 것을 확인할 수 있다.

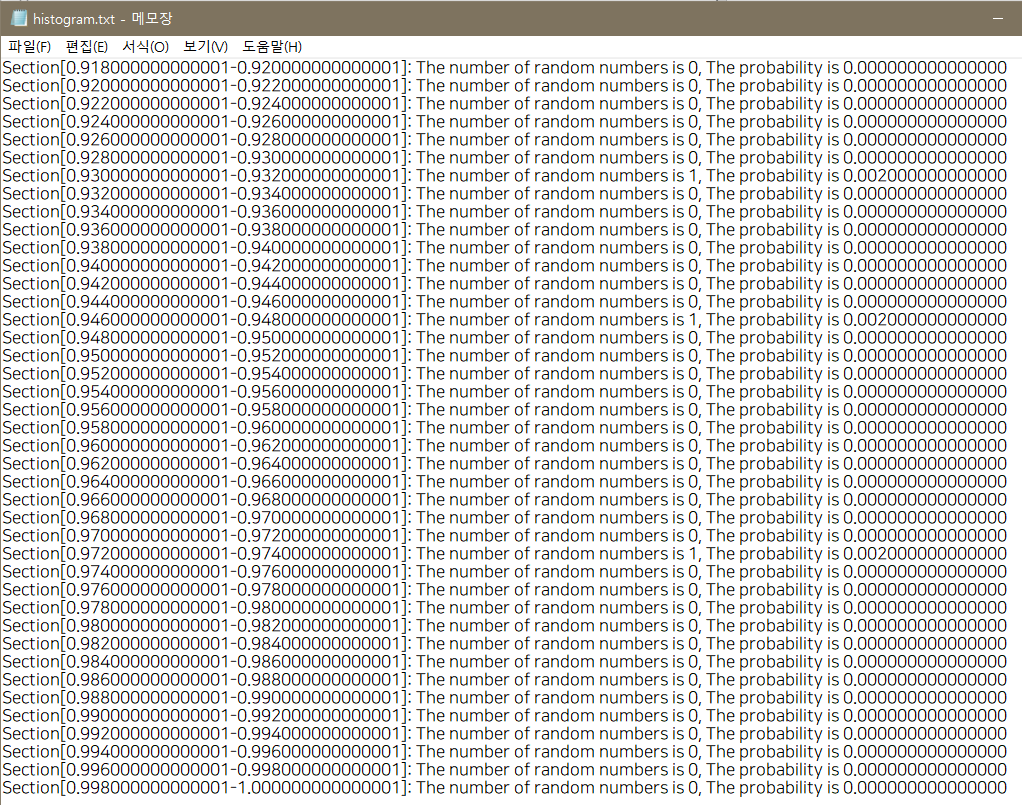
프로그램 2-3을 수행한 결과 histogram.txt는 다음과 같이 출력되었다.



초기구간에는 난수의 개수가 대부분 0인 것을 확인할 수 있다.



커브 그래프와 같이 중간 구간에서는 난수의 개수가 4개에서 9개까지 늘어난 것을 확인할 수 있다.



끝 구간을 확인하면 난수의 개수가 다시 대부분 0으로 줄어든 것을 확인할 수 있다.

histogram.txt를 확인해 본 결과 난수의 개수가 상대적으로 큰 구간은 곡선에서의 함수값이 큰 부분과 대략적으로 일치하고, 상대적으로 작은 구간은 곡선에서의 함수값이 작은 부분과 대략적으로 일치한다. 따라서 발생한 난수의 분포가 원래 곡선에 수렴했다고 볼 수 있으므로 프로그램이 올바르게 작동한다고 판단하였다.

**(iii-iv) 비선형 방정식의 근을 구하는 방법 구현 시 앞에서 사용한 Bisection 방법을 Secant 방법으로 대치한 난수 생성 프로그램을 작성하라. (프로그램 2-2(b)). 물론 자신의 프로그램이 제대로 작동하는지 실험적으로 확인하라. 충분히 큰 난수의 개수 nr에 대해 프로그램 2-2(a)와 프로그램 2-2(b)를 수행시킨 후, 각 방법이 난수를 생성하는 데 걸린 시간을 비교하라.**

각 방법이 제대로 작동하는지 확인하기 위해 Bisection Method, Secant Method, Newton Method세 방법 모두 임시로 출력 결과를 각각 다른 txt 파일로 출력해 일차적으로 확인하고, 각각의 Histogram.txt 출력 결과가 곡선과 대략적으로 일치해 확인할 수 있었다.

**[Secant Method – program2\_2\_b]**

이 방법을 쓸 때 쓰인 함수는 Bisection Method를 구현할 때 썼던 적분 함수 double \_f(double xn, double seed, double\* x, double\* y, int n)를 재사용하였다.

Secant Method는 Bisection Method처럼 초기값 두 개가 필요하다. 초기값 x0과 x1을 0.0, 1.0으로 설정했을 경우 계산 시간은 다음과 같았다. 난수 개수는 1000개로 지정하였다.



Bisection Method에 비해 계산 시간은 적었지만 많이 차이 나지는 않았다.

하지만 초기값 설정을 위해 [0.0, 1.0]을 초기 구간으로 하여 Bisection 방법을 몇 회 반복하여 구간의 폭을 줄인 후, 그 중점을 Secant Method의 초기값으로 설정하게 되면 계산 시간은 다음과 같았다.

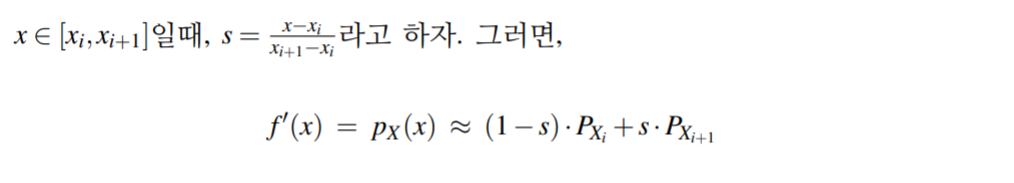


구간의 폭을 줄이니 계산 시간을 줄이는 데 효과적인 것을 확인할 수 있었다. 구간의 폭을 줄이는 과정은 5번-10번이 적당해 보인다. 100회 등과 같이 이 과정을 너무 많이 반복할 경우 오히려 계산 시간이 더 오래 걸릴 수 있기 때문이다.

|  |  |  |  |
| --- | --- | --- | --- |
| **구분** | **유용성** | **정확도** | **비용(시간)** |
| **Bisection** | Win | Draw | Lose |
| **Secant** | Lose | Draw | Win |

**[Newton Method – program2\_2\_b]**

이 방법은 방정식의 함수 f(X)에 대한 함수값뿐만 아니라 미분 값 f’(x)도 계산할 수 있어야 하는데, 선형 보간 방법을 적용하여 미분 함수 \_fp(double xn, double seed, double \*x, double \*y, int n)을 구현하여 사용하였다. 이때 참고한 식은 다음과 같다.



Newton Method를 쓸 때는 Secant Method 부분을 주석 처리하고, 주석 처리되어 있었던 Newton Method 부분을 풀어 사용하였다. 실행 결과 계산 시간은 다음과 같이 걸렸다.



Newton Method 역시 Bisection Method와 비교했을 때 계산 시간이 확연히 더 적게 걸렸다.

|  |  |  |  |
| --- | --- | --- | --- |
| **구분** | **유용성** | **정확도** | **비용(시간)** |
| **Bisection** | Win | Draw | Lose |
| **Newton** | Lose | Draw | Win |